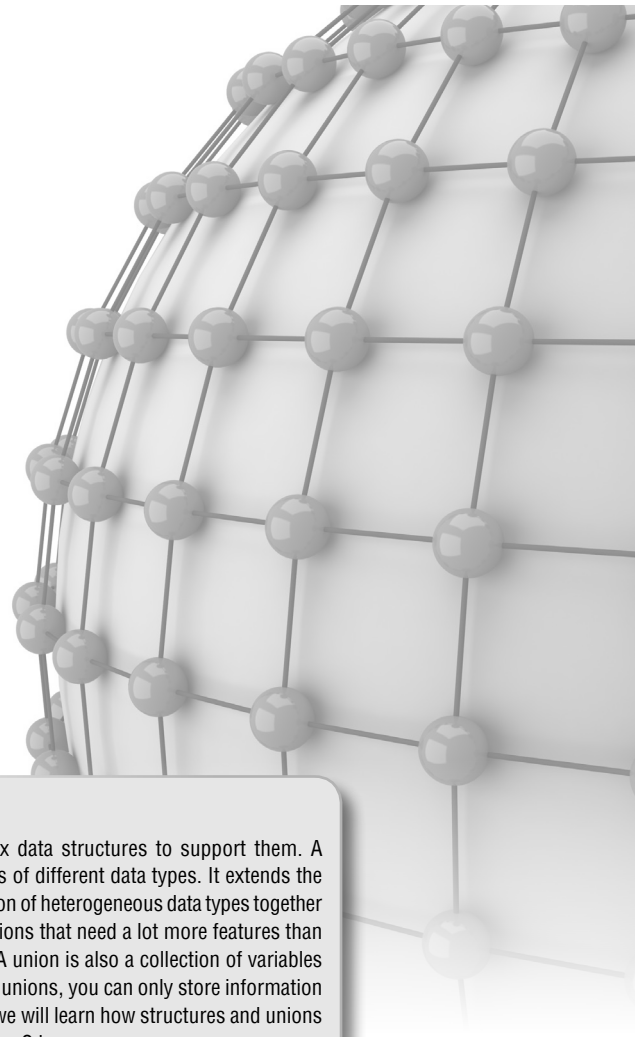
CHAPTER 5

Structures and Unions



LEARNING OBJECTIVE

Today's modern applications need complex data structures to support them. A structure is a collection of related data items of different data types. It extends the concept of arrays by storing related information of heterogeneous data types together under a single name. It is useful for applications that need a lot more features than those provided by the primitive data types. A union is also a collection of variables of different data types, except that in case of unions, you can only store information in one field at any one time. In this chapter, we will learn how structures and unions are declared, defined, and accessed using the C language.

5.1 INTRODUCTION

A structure is in many ways similar to a record. It stores related information about an entity. Structure is basically a user-defined data type that can store related information (even of different data types) together. The major difference between a structure and an array is that an array can store only information of same data type.

A structure is therefore a collection of variables under a single name. The variables within a structure are of different data types and each has a name that is used to select it from the structure.

5.1.1 Structure Declaration

A structure is declared using the keyword `struct` followed by the structure name. All the variables of the structure are declared within the structure. A structure type is generally declared by using the following syntax:

```
struct struct-name
{
      data_type var-name;
```

Programming Tip

Do not forget to place a semicolon after the declaration of structures and unions.

```
    data_type var-name;
    ..............
};
```

For example, if we have to define a structure for a student, then the related information for a student probably would be: `roll_number`, `name`, `course`, and `fees`. This structure can be declared as:

```
struct student
{
      int r_no;
      char name[20];
      char course[20];
      float fees;
};
```

Now the structure has become a user-defined data type. Each variable name declared within a structure is called a member of the structure. The structure declaration, however, does not allocate any memory or consume storage space. It just gives a template that conveys to the C compiler how the structure would be laid out in the memory and also gives the details of member names. Like any other data type, memory is allocated for the structure when we declare a variable of the structure. For example, we can define a variable of `student` by writing:

```
struct student stud1;
```

Here, `struct student` is a data type and `stud1` is a variable. Look at another way of declaring variables. In the following syntax, the variables are declared at the time of structure declaration.

```
struct student
{
   int r_no;
   char name[20];
   char course[20];
   float fees;
} stud1, stud2;
```

In this declaration we declare two variables `stud1` and `stud2` of the structure `student`. So if you want to declare more than one variable of the structure, then separate the variables using a comma. When we declare variables of the structure, separate memory is allocated for each variable. This is shown in Fig. 5.1.

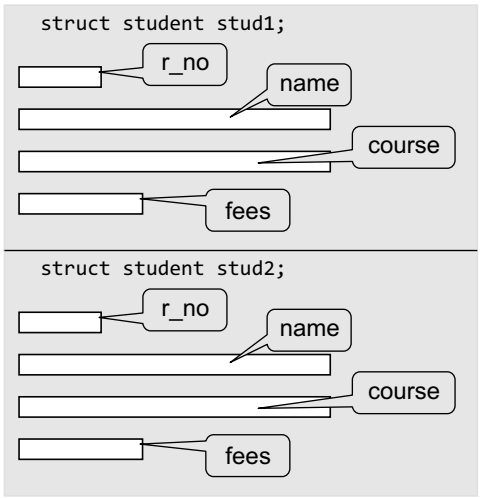


Figure 5.1 Memory allocation for a structure variable

Note Structure type and variable declaration of a structure can be either local or global depending on their placement in the code.

Last but not the least, structure member names and names of the structure follow the same rules as laid down for the names of ordinary variables. However, care should be taken to ensure that the name of structure and the name of a structure member should not be the same. Moreover, structure name and its variable name should also be different.

5.1.2 `Typedef` Declarations

The `typedef` (derived from type definition) keyword enables the programmer to create a new data type name by using an existing data type. By using `typedef`, no new data is created, rather an

alternate name is given to a known data type. The general syntax of using the `typedef` keyword is given as:

```
typedef existing_data_type new_data_type;
```

Programming Tip

C does not allow declaration of variables at the time of creating a typedef definition. So variables must be declared in an independent statement.

Note that `typedef` statement does not occupy any memory; it simply defines a new type. For example, if we write

```
typedef int INTEGER;
```

then `INTEGER` is the new name of data type `int`. To declare variables using the new data type name, precede the variable name with the data type name (new). Therefore, to define an integer variable, we may now write

```
INTEGER num=5;
```

When we precede a `struct` name with the `typedef` keyword, then the `struct` becomes a new type. It is used to make the construct shorter with more meaningful names for types already defined by C or for types that you have declared. For example, consider the following declaration:

```
typedef struct student
{
      int r_no;
      char name[20];
      char course[20];
      float fees;
};
```

Now that you have preceded the structure's name with the `typedef` keyword, `student` becomes a new data type. Therefore, now you can straightaway declare the variables of this new data type as you declare the variables of type `int`, `float`, `char`, `double`, etc. To declare a variable of structure `student`, you may write

```
student stud1;
```

Note that we have not written `struct student stud1`.

5.1.3 Initialization of Structures

A structure can be initialized in the same way as other data types are initialized. Initializing a structure means assigning some constants to the members of the structure. When the user does not explicitly initialize the structure, then C automatically does it. For `int` and `float` members, the values are initialized to zero, and `char` and string members are initialized to `'\0'` by default.

The initializers are enclosed in braces and are separated by commas. However, care must be taken to ensure that the initializers match their corresponding types in the structure definition.

The general syntax to initialize a structure variable is as follows:

```
struct struct_name
{
      data_type member_name1;
      data_type member_name2;
      data_type member_name3;
      .....................
}struct_var = {constant1, constant2, constant3,...};
```

or

```
struct struct_name
{
      data_type member_name1;
      data_type member_name2;
      data_type member_name3;
      .....................
};
```

```
struct struct_name struct_var = {constant1, constant2, constant 3,...};
```

Programming Tip

It is an error to assign a structure of one type to a structure of another type.

For example, we can initialize a student structure by writing,

```
struct student
{
   int r_no;
   char name[20];
   char course[20];
   float fees;
}stud1 = {01, "Rahul", "BCA", 45000};
```

Or, by writing,

```
struct student stud1 = {01, "Rahul", "BCA", 45000};
```

Figure 5.2 illustrates how the values will be assigned to individual fields of the structure.

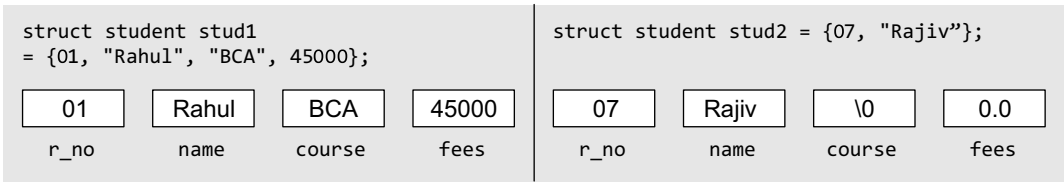


Figure 5.2 Assigning values to structure elements

When all the members of a structure are not initialized, it is called partial initialization. In case of partial initialization, first few members of the structure are initialized and those that are uninitialized are assigned default values.

5.1.4 Accessing the Members of a Structure

Each member of a structure can be used just like a normal variable, but its name will be a bit longer. A structure member variable is generally accessed using a `'.'` (dot) operator. The syntax of accessing a structure or a member of a structure can be given as:

```
struct_var.member_name
```

The dot operator is used to select a particular member of the structure. For example, to assign values to the individual data members of the structure variable `stud1`, we may write

```
stud1.r_no = 01;
stud1.name = "Rahul";
stud1.course = "BCA";
stud1.fees = 45000;
```

To input values for data members of the structure variable `stud1`, we may write

```
scanf("%d", &stud1.r_no);
scanf("%s", stud1.name);
```

Similarly, to print the values of structure variable `stud1`, we may write

```
printf("%s", stud1.course);
printf("%f", stud1.fees);
```

Memory is allocated only when we declare the variables of the structure. In other words, the memory is allocated only when we instantiate the structure. In the absence of any variable, structure definition is just a template that will be used to reserve memory when a variable of type `struct` is declared.

Once the variables of a structure are defined, we can perform a few operations on them. For example, we can use the assignment operator `(=)` to assign the values of one variable to another.

Note Of all the operators –>, . , ( ), and [] have the highest priority. This is evident from the following statement
`stud1.fees++` will be interpreted as `(stud1.fees)++`.

5.1.5 Copying and Comparing Structures

We can assign a structure to another structure of the same type. For example, if we have two structure variables `stud1` and `stud2` of type `struct student` given as

```
struct student stud1 = {01, "Rahul", "BCA", 45000};
struct student stud2;
```

Then to assign one structure variable to another, we will write

```
stud2 = stud1;
```

This statement initializes the members of `stud2` with the values of members of `stud1`. Therefore, now the values of `stud1` and `stud2` can be given as shown in Fig. 5.3.

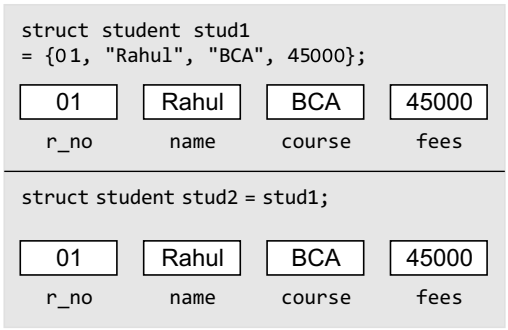


Figure 5.3 Values of structure variables

Programming Tip

An error will be generated if you try to compare two structure variables.

C does not permit comparison of one structure variable with another. However, individual members of one structure can be compared with individual members of another structure. When we compare one structure member with another structure's member, the comparison will behave like any other ordinary variable comparison. For example, to compare the fees of two students, we will write

```
if(stud1.fees > stud2.fees) //to check if fees of stud1 is greater than stud2
```

Programming Examples

1. Write a program using structures to read and display the information about a student.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        struct student
        {
                int roll_no;
                char name[80];
                float fees;
                char DOB[80];
        };
        struct student stud1;
        clrscr();
        printf("\n Enter the roll number : ");
        scanf("%d", &stud1.roll_no);
        printf("\n Enter the name : ");
        scanf("%s", stud1.name);
        printf("\n Enter the fees : ");
        scanf("%f", &stud1.fees);
        printf("\n Enter the DOB : ");
        scanf("%s", stud1.DOB);
        printf("\n ********STUDENT'S DETAILS *******");
        printf("\n ROLL No. = %d", stud1.roll_no);
        printf("\n NAME = %s", stud1.name);
        printf("\n FEES = %f", stud1.fees);
```

```
        printf("\n DOB = %s", stud1.DOB);
        getch();
        return 0;
}
```

Output

```
Enter the roll number : 01
Enter the name : Rahul
Enter the fees : 45000
Enter the DOB : 25-09-1991
********STUDENT'S DETAILS *******
ROLL No. = 01
NAME = Rahul
FEES = 45000.00
DOB = 25-09-1991
```

2. Write a program to read, display, add, and subtract two complex numbers.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        typedef struct complex
        {
                int real;
                int imag;
        }COMPLEX;
        COMPLEX c1, c2, sum_c, sub_c;
        int option;
        clrscr();
        do
        {
                printf("\n ******** MAIN MENU *********");
                printf("\n 1. Read the complex numbers");
                printf("\n 2. Display the complex numbers");
                printf("\n 3. Add the complex numbers");
                printf("\n 4. Subtract the complex numbers");
                printf("\n 5. EXIT");
                printf("\n Enter your option : ");
                scanf("%d", &option);
                switch(option)
                {
                      case 1:
                             printf("\n Enter the real and imaginary parts of the
                             first complex number : ");
                             scanf("%d %d", &c1.real, &c1.imag);
                             printf("\n Enter the real and imaginary parts of the
                             second complex number : ");
                             scanf("%d %d", &c2.real, &c2.imag);
                             break;
                      case 2:
                             printf("\n The first complex number is : %d+%di",
                             c1.real,c1.imag);
                             printf("\n The second complex number is : %d+%di",
                             c2.real,c2.imag);
                             break;
                      case 3:
                             sum_c.real = c1.real + c2.real;
                             sum_c.imag = c1.imag + c2.imag;
                             printf("\n The sum of two complex numbers is :
```

```
                                %d+%di",sum_c.real, sum_c.imag);
                                break;
                        case 4:
                                sub_c.real = c1.real - c2.real;
                                sub_c.imag = c1.imag - c2.imag;
                                printf("\n The difference between two complex numbers
                                is :%d+%di", sub_c.real, sub_c.imag);
                                break;
                        }
            }while(option != 5);
            getch();
            return 0;
    }
```

Output

```
******** MAIN MENU *********
1. Read the complex numbers
2. Display the complex numbers
3. Add the complex numbers
4. Subtract the complex numbers
5. EXIT
Enter your option : 1
Enter the real and imaginary parts of the first complex number : 2 3
Enter the real and imaginary parts of the second complex number : 4 5
Enter your option : 2
The first complex numbers is : 2+3i
The second complex numbers is : 4+5i
Enter your option : 3
The sum of two complex numbers is : 6+8i
Enter your option : 5
```

Because of constraint of space, we will show the MENU only once in all the menu-driven programs.

5.2 NESTED STRUCTURES

A structure can be placed within another structure, i.e., a structure may contain another structure as its member. A structure that contains another structure as its member is called a *nested structure*.

Let us now see how we declare nested structures. Although it is possible to declare a nested structure with one declaration, it is not recommended. The easier and clearer way is to declare the structures separately and then group them in the higher level structure. When you do this, take care to check that nesting must be done from inside out (from lowest level to the most inclusive level), i.e., declare the innermost structure, then the next level structure, working towards the outer (most inclusive) structure.

```
typedef struct
{
      char first_name[20];
      char mid_name[20];
      char last_name[20];
}NAME;
typedef struct
{
      int dd;
      int mm;
      int yy;
}DATE;
typedef struct
{
      int r_no;
```

```
        NAME name;
        char course[20];
        DATE DOB;
        float fees;
} student;
```

In this example, we see that the structure `student` contains two other structures, `NAME` and `DATE`. Both these structures have their own fields. The structure `NAME` has three fields: `first_name`, `mid_name`, and `last_name`. The structure `DATE` also has three fields: `dd`, `mm`, and `yy`, which specify the day, month, and year of the date. Now, to assign values to the structure fields, we will write

```
student stud1;
stud1.r_no = 01;
stud1.name.first_name = "Janak";
stud1.name.mid_name = "Raj";
stud1.name.last_name = "Thareja";
stud1.course = "BCA";
stud1.DOB.dd = 15;
stud1.DOB.mm = 09;
stud1.DOB.yy = 1990;
stud1.fees = 45000;
```

In case of nested structures, we use the dot operator in conjunction with the structure variables to access the members of the innermost as well as the outermost structures. The use of nested structures is illustrated in the next program.

Programming Example

3. Write a program to read and display the information of a student using a nested structure.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        struct DOB
        {
                int day;
                int month;
                int year;
        };
        struct student
        {
                int roll_no;
                char name[100];
                float fees;
                struct DOB date;
        };
        struct student stud1;
        clrscr();
        printf("\n Enter the roll number : ");
        scanf("%d", &stud1.roll_no);
        printf("\n Enter the name : ");
        scanf("%s", stud1.name);
        printf("\n Enter the fees : ");
        scanf("%f", &stud1.fees);
        printf("\n Enter the DOB : ");
        scanf("%d %d %d", &stud1.date.day, &stud1.date.month, &stud1.date.year);
```

```
            printf("\n ********STUDENT'S DETAILS *******");
            printf("\n ROLL No. = %d", stud1.roll_no);
            printf("\n NAME = %s", stud1.name);
            printf("\n FEES = %f", stud1.fees);
            printf("\n DOB = %d – %d – %d", stud1.date.day, stud1.date.month, stud1.date.year);
            getch();
            return 0;
    }
```

Output

```
Enter the roll number : 01
Enter the name : Rahul
Enter the fees : 45000
Enter the DOB : 25 09 1991
********STUDENT'S DETAILS *******
ROLL No. = 01
NAME = Rahul
FEES = 45000.00
DOB = 25 – 09 – 1991
```

5.3 ARRAYS OF STRUCTURES

In the above examples, we have seen how to declare a structure and assign values to its data members. Now, we will discuss how an array of structures is declared. For this purpose, let us first analyse where we would need an array of structures.

In a class, we do not have just one student. But there may be at least 30 students. So, the same definition of the structure can be used for all the 30 students. This would be possible when we make an array of structures. An array of structures is declared in the same way as we declare an array of a built-in data type.

Another example where an array of structures is desirable is in case of an organization. An organization has a number of employees. So, defining a separate structure for every employee is not a viable solution. So, here we can have a common structure definition for all the employees. This can again be done by declaring an array of structure employee.

The general syntax for declaring an array of structures can be given as,

```
struct struct_name
{
      data_type member_name1;
      data_type member_name2;
      data_type member_name3;
      ......................
};
struct struct_name struct_var[index];
```

Consider the given structure definition.

```
struct student
{
      int r_no;
      char name[20];
      char course[20];
      float fees;
};
```

A student array can be declared by writing,

```
struct student stud[30];
```

Now, to assign values to the `ith` student of the class, we will write

```
stud[i].r_no = 09;
stud[i].name = "RASHI";
```

```
stud[i].course = "MCA";
stud[i].fees = 60000;
```

In order to initialize the array of structure variables at the time of declaration, we can write as follows:

```
struct student stud[3] = {{01, "Aman", "BCA", 45000},{02, "Aryan", "BCA", 60000}, {03,
"John", "BCA", 45000}};
```

Programming Example

4. Write a program to read and display the information of all the students in a class. Then edit the details of the ith student and redisplay the entire information.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
        struct student
        {
                int roll_no;
                char name[80];
                int fees;
                char DOB[80];
        };
        struct student stud[50];
        int n, i, num, new_rolno;
        int new_fees;
        char new_DOB[80], new_name[80];
        clrscr();
        printf("\n Enter the number of students : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n Enter the roll number : ");
                scanf("%d", &stud[i].roll_no);
                printf("\n Enter the name : ");
                gets(stud[i].name);
                printf("\n Enter the fees : ");
                scanf("%d",&stud[i].fees);
                printf("\n Enter the DOB : ");
                gets(stud[i].DOB);
        }
        for(i=0;i<n;i++)
        {
                printf("\n ********DETAILS OF STUDENT %d*******", i+1);
                printf("\n ROLL No. = %d", stud[i].roll_no);
                printf("\n NAME = %s", stud[i].name);
                printf("\n FEES = %d", stud[i].fees);
                printf("\n DOB = %s", stud[i].DOB);
        }
        printf("\n Enter the student number whose record has to be edited : ");
        scanf("%d", &num);
        num= num-1;
        printf("\n Enter the new roll number : ");
        scanf("%d", &new_rolno);
        printf("\n Enter the new name : "):
        gets(new_name);
        printf("\n Enter the new fees : ");
```

```
            scanf("%d", &new_fees);
            printf("\n Enter the new DOB : ");
            gets(new_DOB);
            stud[num].roll_no = new_rolno;
            strcpy(stud[num].name, new_name);
            stud[num].fees = new_fees;
            strcpy (stud[num].DOB, new_DOB);
            for(i=0;i<n;i++)
            {
                    printf("\n ********DETAILS OF STUDENT %d*******", i+1);
                    printf("\n ROLL No. = %d", stud[i].roll_no);
                    printf("\n NAME = %s", stud[i].name);
                    printf("\n FEES = %d", stud[i].fees);
                    printf("\n DOB = %s", stud[i].DOB);
            }
            getch();
            return 0;
    }
```

Output

```
Enter the number of students : 2
Enter the roll number : 1
Enter the name : kirti
Enter the fees : 5678
Enter the DOB : 9 9 91
Enter the roll number : 2
Enter the name : kangana
Enter the fees : 5678
Enter the DOB : 27 8 91
********DETAILS OF STUDENT 1*******
ROLL No. = 1
NAME = kirti
FEES = 5678
DOB = 9 9 91
********DETAILS OF STUDENT 2*******
ROLL No. = 2
NAME = kangana
FEES = 5678
DOB = 27 8 91
Enter the student number whose record has to be edited : 2
Enter the new roll number : 2
Enter the new name : kangana khullar
Enter the new fees : 7000
Enter the new DOB : 27 8 92
********DETAILS OF STUDENT 1*******
ROLL No. = 1
NAME = kirti
FEES = 5678
DOB = 9 9 91
********DETAILS OF STUDENT 2*******
ROLL No. = 2
NAME = kangana khullar
FEES = 7000
DOB = 27 8 92
```

5.4 STRUCTURES AND FUNCTIONS

For structures to be fully useful, we must have a mechanism to pass them to functions and return them. A function may access the members of a structure in three ways as shown in Fig. 5.4.

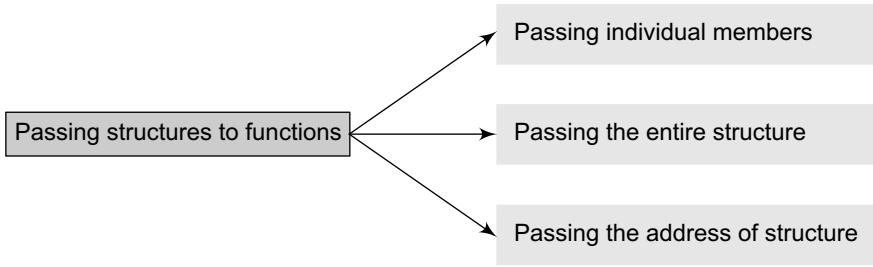


Figure 5.4 Different ways of passing structures to functions

5.4.1 Passing Individual Members

To pass any individual member of a structure to a function, we must use the direct selection operator to refer to the individual members. The called program does not know if a variable is an ordinary variable or a structured member. Look at the code given below which illustrates this concept.

```
#include <stdio.h>
typedef struct
{
      int x;
      int y;
}POINT;
void display(int, int);
int main()
{
      POINT p1 = {2, 3};
      display(p1.x, p1.y);
      return 0;
}
void display(int a, int b)
{
      printf(" The coordinates of the point are: %d %d", a, b);
}
```

Output

```
The coordinates of the point are: 2 3
```

5.4.2 Passing the Entire Structure

Just like any other variable, we can pass an entire structure as a function argument. When a structure is passed as an argument, it is passed using the call by value method, i.e., a copy of each member of the structure is made.

The general syntax for passing a structure to a function and returning a structure can be given as,

```
struct struct_name func_name(struct struct_name struct_var);
```

The above syntax can vary as per the requirement. For example, in some situations, we may want a function to receive a structure but return a void or the value of some other data type. The code given below passes a structure to a function using the call by value method.

```
#include <stdio.h>
typedef struct
{
      int x;
      int y;
```

```
}POINT;
void display(POINT);
int main()
{
      POINT p1 = {2, 3};
      display(p1);
      return 0;
}
void display(POINT p)
{
      printf("The coordinates of the point are: %d %d", p.x, p.y);
}
```

Programming Example

5. Write a program to read, display, add, and subtract two distances. Distance must be defined using kms and meters.

```
#include <stdio.h>
#include <conio.h>
typedef struct distance
{
        int kms;
        int meters;
}DISTANCE;
DISTANCE add_distance (DISTANCE, DISTANCE);
DISTANCE subtract_distance (DISTANCE, DISTANCE);
DISTANCE d1, d2, d3, d4;
int main()
{
        int option;
        clrscr();
        do
        {
                printf("\n ******** MAIN MENU *********");
                printf("\n 1. Read the distances ");
                printf("\n 2. Display the distances");
                printf("\n 3. Add the distances");
                printf("\n 4. Subtract the distances");
                printf("\n 5. EXIT");
                printf("\n Enter your option : ");
                scanf("%d", &option);
                switch(option)
                {
                        case 1:
                             printf("\n Enter the first distance in kms and meters: ");
                             scanf("%d %d", &d1.kms, &d1.meters);
                             printf("\n Enter the second distance in kms and meters: ");
                             scanf("%d %d", &d2.kms, &d2.meters);
                             break;
                        case 2:
                             printf("\n The first distance is : %d kms %d meters",
d1.kms, d1.meters);
                             printf("\n The second distance is : %d kms %d meters",
d2.kms, d2.meters);
                             break;
```

```
                              case 3:
                                   d3 = add_distance(d1, d2);
                                   printf("\n The sum of two distances is : %d kms %d
    meters", d3.kms, d3.meters);
                                   break;
                              case 4:
                                   d4 = subtract_distance(d1, d2);
                                   printf("\n The difference between two distances is : %d
    kms %d meters", d4.kms, d4.meters);
                                   break;
                         }
          }while(option != 5);
          getch();
          return 0;
    }
    DISTANCE add_distance(DISTANCE d1, DISTANCE d2)
    {
          DISTANCE sum;
          sum.meters = d1.meters + d2.meters;
          sum.kms = d1.kms + d2.kms;
          while (sum.meters >= 1000)
          {
                    sum.meters = sum.meters % 1000;
                    sum.kms += 1;
          }
          return sum;
    }
    DISTANCE subtract_distance(DISTANCE d1, DISTANCE d2)
    {
          DISTANCE sub;
          if(d1.kms > d2.kms)
          {
                    sub.meters = d1.meters – d2.meters;
                    sub.kms = d1.kms – d2.kms;
          }
          else
          {
                    sub.meters = d2.meters – d1.meters;
                    sub.kms = d2.kms – d1.kms;
          }
          if(sub.meters < 0)
          {
                    sub.kms = sum.kms – 1;
                    sub.meters = sum.meters + 1000;
          }
          return sub;
    }
```

Output

```
******** MAIN MENU *********
1. Read the distances
2. Display the distances
3. Add the distances
4. Subtract the distances
5. EXIT
```

```
Enter your option : 1
Enter the first distance in kms and meters: 5 300
Enter the second distance in kms and meters: 3 400
Enter your option : 3
The sum of two distances is: 8 kms 700 meters
Enter your option : 5
```

Let us summarize some points that must be considered while passing a structure to the called function.

- If the called function is returning a copy of the entire structure then it must be declared as `struct` followed by the structure name.
- The structure variable used as parameter in the function declaration must be the same as that of the actual argument in the called function (and that should be the name of the `struct` type).
- When a function returns a structure, then in the calling function the returned structure must be assigned to a structure variable of the same type.

5.4.3 Passing Structures through Pointers

Passing large structures to functions using the call by value method is very inefficient. Therefore, it is preferred to pass structures through pointers. It is possible to create a pointer to almost any type in C, including the user-defined types. It is extremely common to create pointers to structures. Like in other cases, a pointer to a structure is never itself a structure, but merely a variable that holds the address of a structure. The syntax to declare a pointer to a structure can be given as,

```
struct struct_name
{
      data_type member_name1;
      data_type member_name2;
      data_type member_name3;
      ......................
}*ptr;
```

or,

```
struct struct_name *ptr;
```

For our `student` structure, we can declare a pointer variable by writing

```
struct student *ptr_stud, stud;
```

The next thing to do is to assign the address of `stud` to the pointer using the address operator (`&`), as we would do in case of any other pointer. So to assign the address, we will write

```
ptr_stud = &stud;
```

To access the members of a structure, we can write

```
/* get the structure, then select a member */
(*ptr_stud).roll_no;
```

Since parentheses have a higher precedence than `*`, writing this statement would work well. But this statement is not easy to work with, especially for a beginner. So, C introduces a new operator to do the same task. This operator is known as 'pointing-to' operator (–>). It can be used as:

```
/* the roll_no in the structure ptr_stud points to */
ptr_stud –> roll_no = 01;
```

This statement is far easier than its alternative.

Programming Tip

The selection operator (–>) is a single token, so do not place any white space between them.

PROGRAMMING EXAMPLES

6. Write a program to initialize the members of a structure by using a pointer to the structure.

```
#include <stdio.h>
#include <conio.h>
struct student
{
        int r_no;
        char name[20];
        char course[20];
        int fees;
};
int main()
{
        struct student stud1, *ptr_stud1;
        clrscr();
        ptr_stud1 = &stud1;
        printf("\n Enter the details of the student :");
        printf("\n Enter the Roll Number =");
        scanf("%d", &ptr_stud1 -> r_no);
        printf("\n Enter the Name = );
        gets(ptr_stud1 -> name);
        printf("\n Enter the Course = ");
        gets(ptr_stud1 -> course);
        printf("\n Enter the Fees = ");
        scanf("%d", &ptr_stud1 -> fees);
        printf("\n DETAILS OF THE STUDENT");
        printf("\n ROLL NUMBER = %d", ptr_stud1 -> r_no);
        printf("\n NAME = %s", ptr_stud1 -> name);
        printf("\n COURSE = %s", ptr_stud1 -> course);
        printf("\n FEES = %d", ptr_stud1 -> fees);
        return 0;
}
```

Output

```
Enter the details of the student:
Enter the Roll Number = 02
Enter the Name = Aditya
Enter the Course = MCA
Enter the Fees = 60000
DETAILS OF THE STUDENT
ROLL NUMBER = 02
NAME = Aditya
COURSE = MCA
FEES = 60000
```

7. Write a program, using an array of pointers to a structure, to read and display the data of students.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
struct student
{
        int r_no;
        char name[20];
        char course[20];
        int fees;
};
struct student *ptr_stud[10];
int main()
```

```
{
        int i, n;
        printf("\n Enter the number of students : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
        ptr_stud[i] = (struct student *)malloc(sizeof(struct student));
        printf("\n Enter the data for student %d ", i+1);
        printf("\n ROLL NO.: ");
        scanf("%d", &ptr_stud[i]->r_no);
        printf("\n NAME: ");
        gets(ptr_stud[i]->name);
        printf("\n COURSE: ");
        gets(ptr_stud[i]->course);
        printf("\n FEES: ");
        scanf("%d", &ptr_stud[i]->fees);
        }
printf("\n DETAILS OF STUDENTS");
for(i=0;i<n;i++)
        {
        printf("\n ROLL NO. = %d", ptr_stud[i]->r_no);
        printf("\n NAME = %s", ptr_stud[i]->name);
        printf("\n COURSE = %s", ptr_stud[i]->course);
        printf("\n FEES = %d", ptr_stud[i]->fees);
        }
return 0;
}
```

Output

```
Enter the number of students : 1
Enter the data for student 1
ROLL NO.: 01
NAME: Rahul
COURSE: BCA
FEES: 45000
DETAILS OF STUDENTS
ROLL NO. = 01
NAME = Rahul
COURSE = BCA
FEES = 45000
```

8. Write a program that passes a pointer to a structure to a function.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
struct student
{
        int r_no;
        char name[20];
        char course[20];
        int fees;
};
void display (struct student *);
int main()
{
        struct student *ptr;
        ptr = (struct student *)malloc(sizeof(struct student));
        printf("\n Enter the data for the student ");
        printf("\n ROLL NO.: ");
        scanf("%d", &ptr->r_no);
```

```
            printf("\n NAME: ");
            gets(ptr->name);
            printf("\n COURSE: ");
            gets(ptr->course);
            printf("\n FEES: ");
            scanf("%d", &ptr->fees);
            display(ptr);
            getch();
            return 0;
    }
    void display(struct student *ptr)
    {
            printf("\n DETAILS OF STUDENT");
            printf("\n ROLL NO. = %d", ptr->r_no);
            printf("\n NAME = %s", ptr->name);
            printf("\n COURSE = %s ", ptr->course);
            printf("\n FEES = %d", ptr->fees);
    }
```

Output

```
Enter the data for the student
ROLL NO.: 01
NAME: Rahul
COURSE: BCA
FEES: 45000
DETAILS OF STUDENT
ROLL NO. = 01
NAME = Rahul
COURSE = BCA
FEES = 45000
```

5.5 SELF-REFERENTIAL STRUCTURES

Self-referential structures are those structures that contain a reference to the data of its same type. That is, a self-referential structure, in addition to other data, contains a pointer to a data that is of the same type as that of the structure. For example, consider the structure node given below.

```
struct node
    {
          int val;
          struct node *next;
    };
```

Here, the structure node will contain two types of data: an integer `val` and a pointer `next`. You must be wondering why we need such a structure. Actually, self-referential structure is the foundation of other data structures. We will be using them throughout this book and their purpose will be clearer to you when we discuss linked lists, trees, and graphs.

5.6 UNIONS

Similar to structures, a union is a collection of variables of different data types. The only difference between a structure and a union is that in case of unions, you can only store information in one field at any one time. To better understand a union, think of it as a chunk of memory that is used to store variables of different types. When a new value is assigned to a field, the existing data is replaced with the new data.

Thus, unions are used to save memory. They are useful for applications that involve multiple members, where values need not be assigned to all the members at any one time.

Programming Tip

It is an error to use a structure/ union variable as a member of its own struct type structure or union type union, respectively.

5.6.1 Declaring a Union

The syntax for declaring a union is the same as that of declaring a structure. The only difference is that instead of using the keyword `struct`, the keyword `union` would be used. The syntax for union declaration can be given as

```
union union-name
{
 data_type var-name;
 data_type var-name;
 ..................
};
```

Programming Tip

Variable of a structure or a union can be declared at the time of structure/union definition by placing the variable name after the closing brace and before the semicolon.

Again the `typedef` keyword can be used to simplify the declaration of union variables. The most important thing to remember about a union is that the size of a union is the size of its largest field. This is because sufficient number of bytes must be reserved to store the largest sized field.

5.6.2 Accessing a Member of a Union

A member of a union can be accessed using the same syntax as that of a structure. To access the fields of a union, use the dot operator (.), i.e., the union variable name followed by the dot operator followed by the member name.

5.6.3 Initializing Unions

The difference between a structure and a union is that in case of a union, the fields share the same memory space, so new data replaces any existing data. Look at the following code and observe the difference between a structure and union when their fields are to be initialized.

```
#include <stdio.h>
typedef struct POINT1
{
      int x, y;
};
typedef union POINT2
{
      int x;
      int y;
};
int main()
{
      POINT1 P1 = {2,3};
      // POINT2 P2 ={4,5}; Illegal in case of unions
      POINT2 P2;
      P2.x = 4;
      P2.y = 5;
      printf("\n The coordinates of P1 are %d and %d", P1.x, P1.y);
      printf("\n The coordinates of P2 are %d and %d", P2.x, P2.y);
      return 0;
}
```

Output

```
The coordinates of P1 are 2 and 3
```

```
The coordinates of P2 are 5 and 5
```

In this code, `POINT1` is a structure name and `POINT2` is a union name. However, both the declarations are almost same (except the keywords—`struct` and `union`). In `main()`, we can see the difference between structures and unions while initializing values. The fields of a union cannot be initialized all at once.

Programming Tip

The size of a union is equal to the size of its largest member.

Look at the output carefully. For the structure variable the output is as expected but for the union variable the answer does not seem to be correct. To understand the concept of union, execute the following code. The code given below just re-arranges the `printf` statements. You will be surprised to see the result.

```
#include <stdio.h>
typedef struct POINT1
{
      int x, y;
};
typedef union POINT2
{
      int x;
      int y;
};
int main()
{
      POINT1 P1 = {2,3};
      POINT2 P2;
      printf("\n The coordinates of P1 are %d and %d", P1.x, P1.y);
      P2. x = 4;
      printf("\n The x coordinate of P2 is %d", P2.x);
      P2.y = 5;
      printf("\n The y coordinate of P2 is %d", P2.y);
      return 0;
}
```

Output

```
The coordinates of P1 are 2 and 3
The x coordinate of P2 is 4
The y coordinate of P2 is 5
```

Here although the output is correct, the data is still overwritten in memory.

5.7 ARRAYS OF UNION VARIABLES

Like structures we can also have an array of union variables. However, because of the problem of new data overwriting existing data in the other fields, the program may not display the accurate results.

```
#include <stdio.h>
union POINT
{
      int x, y;
};
int main()
{
      int i;
      union POINT points[3];
      points[0].x = 2;
      points[0].y = 3;
      points[1].x = 4;
      points[1].y = 5;
```

```
        points[2].x = 6;
        points[2].y = 7;
        for(i=0;i<3;i++)
              printf("\n Coordinates of Point[%d] are %d and %d", i, points[i].x,
              points[i].y);
        return 0;
}
```

Output

```
Coordinates of Point[0] are 3 and 3
Coordinates of Point[1] are 5 and 5
Coordinates of Point[2] are 7 and 7
```

5.8 UNIONS INSIDE STRUCTURES

Generally, unions can be very useful when declared inside a structure. Consider an example in which you want a field of a structure to contain a string or an integer, depending on what the user specifies. The following code illustrates such a scenario:

```
#include <stdio.h>
struct student
{
      union
      {
            char name[20];
            int roll_no;
      };
      int marks;
};
int main()
{
      struct student stud;
      char choice;
      printf("\n You can enter the name or roll number of the student");
      printf("\n Do you want to enter the name? (Y or N): ");
      gets(choice);
      if(choice==‘y’ || choice==‘Y’)
      {
            printf("\n Enter the name: ");
            gets(stud.name);
      }
      else
      {
            printf("\n Enter the roll number: ");
            scanf("%d", &stud.roll_no);
      }
      printf("\n Enter the marks: ");
      scanf("%d", &stud.marks);
      if(choice==‘y’ || choice==‘Y’)
            printf("\n Name: %s ", stud.name);
      else
            printf("\n Roll Number: %d ", stud.roll_no);
      printf("\n Marks: %d", stud.marks);
      return 0;
}
```

Now in this code, we have a union embedded within a structure. We know the fields of a union will share memory, so in the main program we ask the user which data he/she would like to store and depending on his/her choice the appropriate field is used.

Points to Remember

- Structure is a user-defined data type that can store related information (even of different data types) together.
- A structure is declared using the keyword `struct`, followed by the structure name.
- The structure definition does not allocate any memory or consume storage space. It just gives a template that conveys to the C compiler how the structure is laid out in the memory and gives details of the member names. Like any data type, memory is allocated for the structure when we declare a variable of the structure.
- When a `struct` name is preceded with the keyword `typedef`, then the `struct` becomes a new type.
- When the user does not explicitly initialize the structure, then C automatically does it. For `int` and `float` members, the values are initialized to zero and `char` and string members are initialized to `'\0'` by default.
- A structure member variable is generally accessed using a `'.'` (dot) operator.
- A structure can be placed within another structure. That is, a structure may contain another structure as its member. Such a structure is called a nested structure.
- Self-referential structures are those structures that contain a reference to data of its same type. That is, a self-referential structure, in addition to other data, contains a pointer to a data that is of the same type as that of the structure.
- A union is a collection of variables of different data types in which memory is shared among these variables. The size of a union is equal to the size of its largest member.
- The only difference between a structure and a union is that in case of unions information can only be stored in one member at a time.

Exercises

Review Questions

1. What is the advantage of using structures?
2. Structure declaration reserves memory for the structure. Comment on this statement with valid justifications.
3. Differentiate between a structure and an array.
4. Write a short note on structures and inter-process communication.
5. Explain the utility of the keyword `typedef` in structures.
6. Explain with an example how structures are initialized.
7. Is it possible to create an array of structures? Explain with the help of an example.
8. What do you understand by a union?
9. Differentiate between a structure and a union.
10. How is a structure name different from a structure variable?
11. Explain how members of a union are accessed.
12. Write a short note on nested structures.
13. In which applications unions can be useful?

Programming Exercises

1. Declare a structure that represents the following hierarchical information.
   (a) Student
   (b) Roll Number
   (c) Name
      (i) First name
      (ii) Middle Name
      (iii) Last Name
   (d) Sex
   (e) Date of Birth
      (i) Day
      (ii) Month
      (iii) Year
   (f) Marks
      (i) English
      (ii) Mathematics
      (iii) Computer Science
2. Define a structure to store the name, an array `marks[]` which stores the marks of three different subjects, and a character grade. Write a program to display the details of the student whose name is entered by the user. Use the structure definition of the first question to make an array of students.

Display the name of the students who have secured less than 40% of the aggregate.

3. Modify Question 2 to print each student's average marks and the class average (that includes average of all the student's marks).
4. Make an array of students as illustrated in Question 1 and write a program to display the details of the student with the given Date of Birth.
5. Write a program to find smallest of three numbers using structures.
6. Write a program to calculate the distance between the given points (6,3) and (2,2).
7. Write a program to read and display the information about all the employees in a department. Edit the details of the i^{th} employee and redisplay the information.
8. Write a program to add and subtract height 6'2" and 5'4".
9. Write a program to add and subtract 10hrs 20mins 50sec and 5hrs 30min 40sec.
10. Write a program using structure to check if the current year is leap year or not.
11. Write a program using pointer to structure to initialize the members of an employee structure. Use functions to print the employee's information.
12. Write a program to create a structure with the information given below. Then, read and print the data.

    Employee[10]
    - (a) Emp_Id
    - (b) Name
      - (i) First Name
      - (ii) Middle Name
      - (iii) Last Name
    - (c) Address
      - (i) Area
      - (ii) City
      - (iii) State
    - (d) Age
    - (e) Salary
    - (f) Designation
13. Define a structure date containing three integers—day, month, and year. Write a program using functions to read data, to validate the date entered by the user and then print the date on the screen. For example, if you enter 29,2,2010 then that is an invalid date as 2010 is not a leap year. Similarly 31,6,2007 is invalid as June does not have 31 days.
14. Using the structure definition of the above program, write a function to increment the date. Make sure that the incremented date is a valid date.
15. Modify the above program to add a specific number of days to the given date.
16. Write a program to define a structure vector. Then write functions to read data, print data, add two vectors and scale the members of a vector by a factor of 10.
17. Write a program to define a structure for a hotel that has members— name, address, grade, number of rooms, and room charges. Write a function to print the names of hotels in a particular grade. Also write a function to print names of hotels that have room charges less than the specified value.
18. Write a program to define a union and a structure both having exactly the same members. Using the `sizeof` operator, print the size of structure variable as well as union variable and comment on the result.
19. Declare a structure time that has three fields—hr, min, sec. Create two variables start_time and end_time. Input their values from the user. Then while start_time does not reach the end_time, display GOOD DAY on the screen.
20. Declare a structure fraction that has two fields—numerator and denominator. Create two variables and compare them using function. Return 0 if the two fractions are equal, –1 if the first fraction is less than the second and 1 otherwise. You may convert a fraction into a floating point number for your convenience.
21. Declare a structure POINT. Input the coordinates of a point variable and determine the quadrant in which it lies. The following table can be used to determine the quadrant

Quadrant	X	Y
1	Positive	Positive
2	Negative	Positive
3	Negative	Negative
4	Positive	Negative

22. Write a program to calculate the area of one of the geometric figures—circle, rectangle or a triangle. Write a function to calculate the area.

The function must receive one parameter which is a structure that contains the type of figure and the size of the components needed to calculate the area must be a part of a union. Note that a circle requires just one component, rectangle requires two components and a triangle requires the size of three components to calculate the area.

Multiple-choice Questions

1. A data structure that can store related information together is called
   (a) Array (b) String
   (c) Structure (d) All of these
2. A data structure that can store related information of different data types together is called
   (a) Array (b) String
   (c) Structure (d) All of these
3. Memory for a structure is allocated at the time of
   (a) Structure definition
   (b) Structure variable declaration
   (c) Structure declaration
   (d) Function declaration
4. A structure member variable is generally accessed using
   (a) Address operator (b) Dot operator
   (c) Comma operator (d) Ternary operator
5. A structure that can be placed within another structure is known as
   (a) Self-referential structure
   (b) Nested structure
   (c) Parallel structure
   (d) Pointer to structure
6. A union member variable is generally accessed using the
   (a) Address operator (b) Dot operator
   (c) Comma operator (d) Ternary operator
7. `typedef` can be used with which of these data types?
   (a) struct (b) union
   (c) enum (d) all of these

True or False

1. Structures contain related information of the same data type.
2. Structure declaration reserves memory for the structure.
3. When the user does not explicitly initialize the structure, then C automatically does it.
4. The dereference operator is used to select a particular member of the structure.
5. A nested structure contains another structure as its member.
6. A struct type is a primitive data type.
7. C permits copying of one structure variable to another.
8. Unions and structures are initialized in the same way.
9. A structure cannot have a union as its member.
10. C permits nested unions.
11. A field in a structure can itself be a structure.
12. No two members of a union should have the same name.
13. A union can have another union as its member.
14. New variables can be created using the typedef keyword.

Fill in the Blanks

1. Structure is a _____ data type.
2. _____ is just a template that will be used to reserve memory when a variable of type `struct` is declared.
3. A structure is declared using the keyword `struct` followed by a _____.
4. When we precede a `struct` name with _____, then the `struct` becomes a new type.
5. For `int` and `float` structure members, the values are initialized to _____.
6. `char` and `string` structure members are initialized to _____ by default.
7. A structure member variable is generally accessed using a _____.
8. A structure placed within another structure is called a _____.
9. _____ structures contain a reference to data of its same type.
10. Memory is allocated for a structure when _____ is done.
11. _____ is a collection of data under one name in which memory is shared among the members.
12. The selection operator is used to _____.
13. _____ permits sharing of memory among different types of data.